CLAIMS

We claim:

1. A method for allowing communication between a Practical Extraction Report Language (PERL) program and a distributed object, comprising the steps of:

a) receiving a request from said PERL program, said request specifying said distributed object;

b) translating said request from said PERL program to a format which is suitable for use with a Common Object Request Broker Architecture (CORBA);

c) making a call to access said distributed object via the Common Object Request Broker Architecture (CORBA);

d) receiving a response from said call in said step c);

e) translating said response to a form which is substantially compliant with the Practical Extraction Report Language; and

f) passing said translated response from said step e) to said PERL program.

2. The method of Claim 1 wherein said step b) comprises the step of:

b1) an adapter program converting a data structure specified by said PERL request into a form which is substantially compliant with a communication program.

3. The method of Claim 2 wherein said communication program comprises a client stub.

4. The method of Claim 1 wherein said step b) comprises the step of:

b1) an adapter program converting said PERL request into a request which is substantially compliant with the Common Object Request Broker Architecture (CORBA) format.

5. The method of Claim 4 wherein said adapter program is written in a first programming language and said PERL application is written in second programming language, said first and said second programming languages being different.

6. The method of Claim 4 wherein said adapter program is substantially compliant with the C programming language.

7. The method of Claim 1 wherein said PERL program is located on a first computer system and said distributed object is located on a second computer system.

8. The method of Claim 1 wherein said step e) comprises the step of:

e1) an adapter program converting a data structure into a form which is substantially compliant with the Practical Extraction Report Language.

9. The method of Claim 1 wherein said step e) comprises the step of:

e1) for a plurality of objects described in an Interface Definition Language (IDL), providing a corresponding plurality of translations in an adapter program,

wherein said adapter program translates between a communication program and said PERL program.

10. The method of Claim 1 further comprising the step of:

g) said PERL program accessing user information over a number of databases by connecting to a server via said CORBA.

11. A computer readable medium having stored thereon program instructions for allowing a Practical Extraction Report Language (PERL) program to communicate with a distributed object via Common Object Request Broker Architecture (CORBA), said instructions carrying out a method comprising the steps of:

a) receiving a request from said PERL program, said request specifying said distributed object;

b) translating said request from said PERL program to a format which is suitable for use with a Common Object Request Broker Architecture (CORBA);

c) making a call to access said distributed object via the Common Object Request Broker Architecture (CORBA);

d) receiving a response from said call in said step c);

e) translating said response to a form which is substantially compliant with the Practical Extraction Report Language; and

f) passing said translated response from said step e) to said PERL program.

12. The computer readable medium of Claim 11 wherein said step b) of said method comprises the step of:

b1) converting said PERL request into a request which is substantially compliant with the Common Object Request Broker Architecture (CORBA) format.

13. The computer readable medium of Claim 11 having further stored therein said PERL program; and

wherein said distributed object is located on a remote computer system.

14. The computer readable medium of Claim 11 wherein said program comprises a client stub.

15. The computer readable medium of Claim 14 wherein said step b) of said method comprises the step of:

b1) converting a data structure into a form which is substantially compliant with the data structures of said client stub.

16. The computer readable medium of Claim 11 having further stored therein said PERL program and said distributed object.

17. The computer readable medium of Claim 11 wherein said step e) of said method comprises the step of:

e1) converting a data structure into a form which is substantially compliant with the Practical Extraction Report Language.

18. The computer readable medium of Claim 11 wherein said step e) of said method comprises the step of:

e1) for a plurality of objects described in an Interface Definition Language (IDL), providing a corresponding plurality of translations.

19. The computer readable medium of Claim 11 wherein said program comprises a module generated by Practical Extraction Report Language External Subroutine (PERL-XS).

20. In a computer system, means for providing communication between a Practical Extraction Report Language (PERL) program and a distributed object comprising:

a) means for translating a call from said PERL program to a format substantially compliant with a Common Object Request Broker Architecture (CORBA); and

b) means for translating a response from said call to a format substantially compliant with the Practical Extraction Report Language.

21. The means for providing communication of Claim 20 further comprising means to access said distributed object via Common Object Request Broker Architecture (CORBA).

22. The means for providing communication of Claim 20 wherein said means for translating said call from said PERL program comprises:

means for converting a data structure into a form which is substantially compliant with a program which accesses said distributed object via said Common Object Request Broker Architecture (CORBA).

23. The means for providing communication of Claim 20 wherein said means for translating said call from said PERL program comprises:

means for converting said PERL request into a request which is substantially compliant with the Common Object Request Broker Architecture (CORBA) format.

Appendix A

```
++++++++ Perl Program Start++++++++++++++++++++++++++++++

# Before 'make install' is performed this script should be runnable with
# 'make test'. After 'make install' it should work as 'perl test.pl'

###############We start with some code to print on failure.

# Change 1.1 below to 1..last_test_to_print.

BEGIN { $1 = 1; print "1..1\n"; }
END {print "not ok 1\n" unless $loaded;}

#### This package performs important work.  Use h2xs utility to develop
###   this package
#### Look for man pages "perlxs" and "perlxstut" for instructions
####  use Mytest2;

$loaded = 1;
print "ok 1\n";

######################## End of print on failure code

#Insert your test code below

# This is a list of fields for which values are to be fetched from distributed
# server object
@fields = ( "view1.field1", "view1.field2", "view1.field3" );

# This is an empty list in which the values for above mentioned fields will
# be returned
@retfields = ();

# One has to call the getValues function provided by Mytest2 package to
# access distributed object
# Mytest2 is a perl module which in turn connects to a dynamic shared
# library which has implemented
# the getValues function

$ret = Mytest2::getValues(\@fields, \@retfields);

# print the values of the fields
# instead of printing these fields, they can be used in Perl programs just
# like other data

foreach my $rf ( @retfields ) {
  print "Output:  $rf \n"
}
+++++++++ Perl Program End +++++++++++++++++++++++++
```

+Dynamic shared library - - Implementation of getValues function Start +

```
int
getValues (in, out)
SV* in;
SV* out;
{
int reqlen = -1, i, no_ele_ret = 0;

AV* input;
AV* output;
SV* strs = SvRV(in); /* de-mangle the input array reference */
SV*rets = SvRV (out); /* de-mangle the output array reference */
SV*temp;

char** reqFields = NULL;  /* Requested fields reference */
char** returnFields = NULL; /*Returned fields reference */
int retval;

if(!SvROK(in)) /* earlier it was "in" */
   croak ("samir: not a reference SV*");
if(SvTYPE(strs) ! = SVt_PVAV )
   croak ("samir:: not a reference to an array");
input = (AV*)strs; /* finally got the reference */

if(!SvROK(out)) /* earlier it was */
   croak ("samir: not a reference SV*");
if(SvTYPE(strs) ! = SVt_PVAV )
   croak ("samir:: not a reference to an array");
output = (AV*)rets; /* finally got the reference */

/* convert the input to a C style input */
reqlen = av_len(input) + 1;

reqFields = (char**) malloc (sizeof(char*) *reqlen);
for (i=0; i <reqlen; i ++ )
{
STRLEN stringLength;
SV** stringHandle = av_fetch(input, i, 0 );
char*stringValue = SvPV("stringHandle, stringLength );
reqFields[i] = stringValue;
}

/* At this point we have all the requested field names accessible through
reqFields reference */
/* We need to get the returned values from a distributed object and make
returnFields point to it */
```

```
/* Convert the requested field list into corba equivalent list */

// Initialize the ORB.
CORBA::ORB_ptr orb = CORBA::ORB_init(argc, argv);

// Locate the distributed server object
upr:: UPR_Record_var svrOhj =
upr::UPR_Record::_bind("DistributeServerObjectName");


// Call server object with appropriate parameters
// Here the parameters to the getValues functions could be input and /or
// output parameters
(ReturnedValue) = svrObj->getValues("Appropriate params for server object
including request fields");

/* Convert the "ReturnedValue" list into a format that could be accessed
using returnFields reference */

return retval; // return the appropriate value as expected
}

++ Dynamic shared library - - Implementation of getValues function End +++

+++++++++ Sample CORBA IDL Definition Start ++++++++++

module upr
{
    typedef sequence <string> string_list;

    interface UPR_Record
    {
        // General Get/Set API (Strings/Lists)
        string_list getValues(
        in string appName,
        in string_list fieldNames );
    };

  };

// ********
// Use this IDL definition to generate the client stubs for above program to
// use.
// Any CORBA implementation vendor's software can be used to do this.
// *******

+++++++++ Sample CORBA IDL Definition End +++++++++
```